CMPSC-265
Data Structures and Algorithms

Zaihan Yang
zyang13@suffolk.edu

Department of Math and Computer Science
Suffolk University

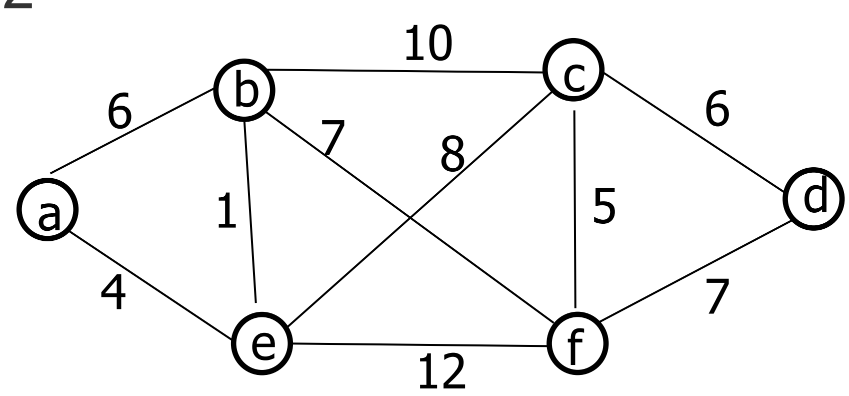
Fall 2019

Review

- Finding Minimum Spanning Tree

The Shortest Path Problem

- One of the main problems defined on weighted graphs with many applications (like network routing).
- We want to find the shortest path between nodes.
  - Shortest means lowest cost, shortest distance,…
- Example: Shortest path from a to f is a->e->b->f with cost/ distance 12



Finding shortest paths

- Single-source shortest paths finding
  - Dijkstra’s algorithm
  - Bellman-Ford algorithm
  - Finding single-source shortest paths on a DAG (directed acyclic graph) using Bellman-Ford algorithm and topological sorting
- All-pairs shortest paths finding
  - Floyd-Warshall algorithm

The Single Source Shortest Path Problem

- Starting from a node, find shortest paths to all other nodes.
  - Finding shortest path between two specific nodes is as complex as finding shortest paths from a source node to all other nodes.

- Dijkstra's algorithm is a *greedy* algorithm to solve the single source shortest path problem.
  - It is similar to the Prim's algorithm.
  - At every step, adds an unvisited node (not in the shortest path tree yet) which has the shortest distance from the source.
  - It then updates the distance from source to neighbors of the newly added node.

Dijkstra’s Algorithm

S = source node, T = visited set, D(v) = (shortest) distance to v, w(i,j) = weight of edge ij

//Initialization

T = {S}

For every vertex v:

  if v is adjacent to S, then D(v) = w(S,v)

  Else D(v) = ∞

While there are vertices left (not in T):

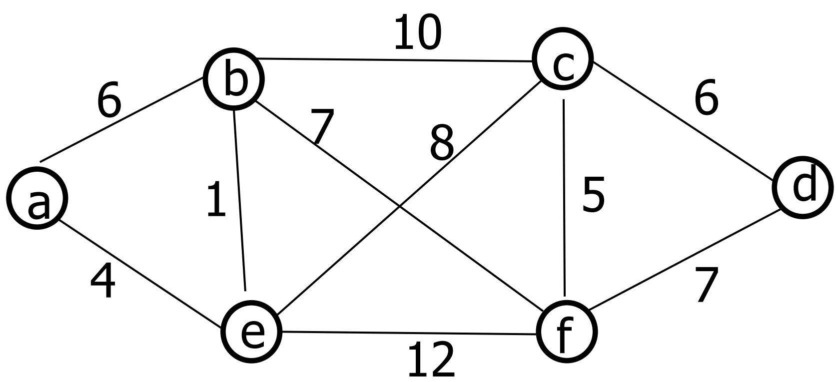
  Find a node u such that u is not in T and D(u) is minimum

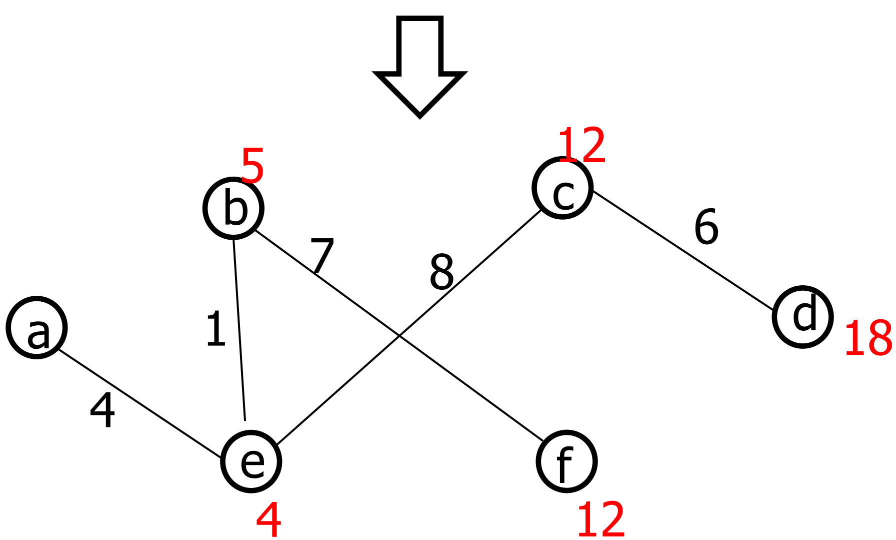
  Add u to T

  for every neighbor v of u which is not in T:

    update D(v) as D(v) = min[D(v), D(u) + w(u,v)]

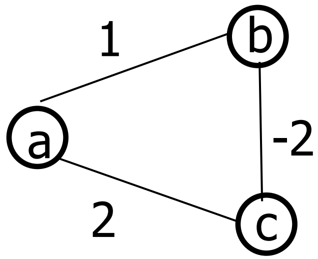
Dijkstra's Algorithm-Example





Comments on Dijkstra's Algorithm

- It is based on the intuition that adding edges will not make distance shorter. Is it optimal?
- Would it work if edges have negative weights?
  - No, consider this example:



    - Dijkstra find shortest path from a to b as a->b with cost of 1, but if we take a->c->b we get there with cost of 0.
  - What is wrong? Why does this happen?
    - That's because we do greedy, we finalize a node without considering other possible paths.

Comments on Single Source Shortest Paths

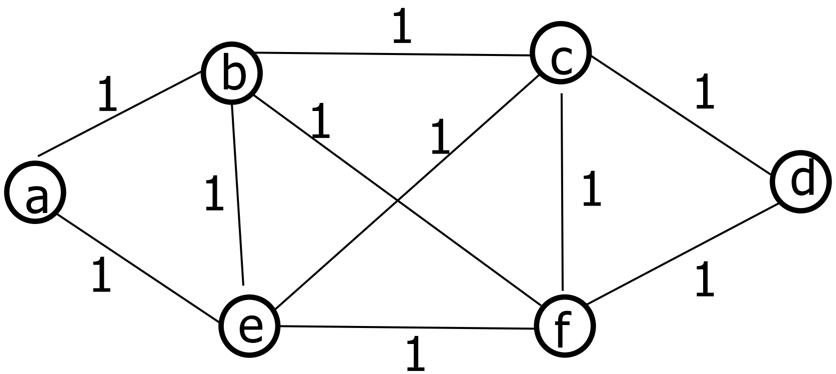
- If you add a constant positive to edge weights to get rid of negative weights, would it solve the problem in Dijkstra's algorithm?
  - It changes the graph and it results in different paths that are not shortest ones in the original graph.

- Dijkstra's algorithm works on directed or undirected graphs with no negative edge weights.

- Time complexity of Dijkstra's algorithm?
  - It is the same as Prim's algorithm.

Special Case of Single Source Shortest Paths

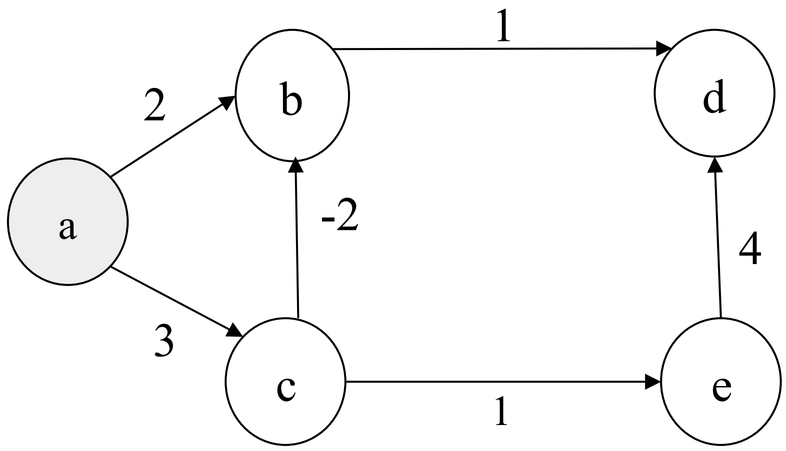
- A special case when all edge weights are identical.
- Just do BFS, and keep track of the previous node.
- This is faster than Dijkstra's algorithm.
- How about DFS?



Bellman-Ford Shortest Path Algorithm

- Bellman-Ford algorithm works with negative-weight edges:
  - Based on *Dynamic Programming* instead of Greedy
    - In dynamic programming you save results of sub-problems in a table to be re-used (instead of re-computing them). Example, store results of Fibonnacci numbers in an array instead of making recursive calls.
  - Use a table (one-dimensional array of size N) to save distance to each node, initialize with infinity except for the source which is 0 distance from itself.
  - N-1 times:
    - look at all edges and see if you need to update distance from source (update table). For edge u->v do: D(v) = min[D(v), D(u) + w(u,v)]
  - At iteration i, it finds shortest paths with at most i edges. So, it is enough to iterate N-1 times, since a shortest path cannot have more than N-1 edges.

Bellman-Ford Shortest Path Algorithm - Example



d[]

a	b	c	d	e

parent[]

a	b	c	d	e

Comments on Bellman-Ford Algorithm

- Works with negative-weight edges.
- O(E*N), Not as fast as Dijkstra's algorithm.
- What if you have negative-weight cycles?
  - Bellman-Ford detects them. How?
    - Do one more iteration(after that N-1 iterations), if you see an improvement in updating distances, it means there is a negative-weight cycle.
  - Ok, but what can you do then?
    - Nothing, maybe just report that you found it.

Bellman-Ford Shortest Path Algorithm

Pseudo-code for Bellman-Ford:

```
Initialize the tables/vectors d and parent
for i := 1 to N - 1 do
     for each edge (u, v) ∈ E do
            if d[v] > d[u] + w(u, v)
                   then  d[v] := d[u] + w(u, v)
                         parent[v] := u

for each edge (u, v) ∈ E do
            if d[v] > d[u] + w(u, v)
                   then return false  // there is a negative cycle
return true
```

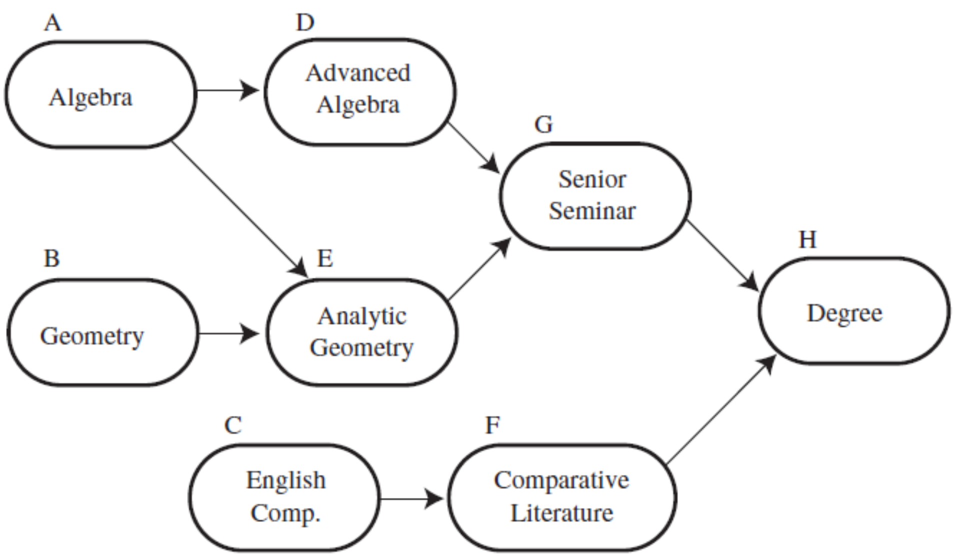
Single-Source Shortest Path in a DAG

- Given a Directed Acyclic Graph (DAG), how do you find the shortest paths efficiently? Can we do better than bellman-ford?
  - Yes, when there is no cycle, we can be done in one pass if we have a *proper ordering* to go over edges.
  - We can make use of topological sort/ordering

- The idea is to go over the edges based on topological ordering and do the same update of distance values as in bellman-ford.
- Time complexity?

Topological Sorting in Directed Graphs

- Nodes must be traversed in a specific order.
- Example: course prerequisites
  - ABCDEFGH is one possible way to get a degree, satisfying prerequisites.



Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 650

Topological Sorting in Directed Graphs

- Three steps:
  - Find a vertex that has no successors (the successors to a vertex are those vertices that are directly downstream from it connected by a directed edge).
  - Delete this vertex from the graph, and insert its label at the beginning of the list (insertFirst() in LinkedList).
  - Repeat steps 1 and 2 until all vertices are gone. At this point, the list shows the vertices arranged in topological order.

- What if there is a cycle?
  - It does not work. It works on Directed Acyclic Graphs (DAG).

Topological Sorting in Directed Graphs

```
public void topo() // topological sort
{
  int orig_nVerts = nVerts; // remember how many verts
  while(nVerts > 0) // while vertices remain,
  {
   // get a vertex with no successors, or -1
   int currentVertex = noSuccessors();
   if(currentVertex == -1) // must be a cycle
    {
      System.out.println("ERROR: Graph has cycles");
      return;
     }
   // insert vertex label in topo-sorted array (start at end)
   sortedArray[nVerts-1] = vertexList[currentVertex].label;
   deleteVertex(currentVertex); // delete vertex
  } // end while

 // vertices all gone; display sortedArray
 System.out.print("Topologically sorted order: ");
 for(int j=0; j<orig_nVerts; j++)
    System.out.print( sortedArray[j] );
 System.out.println("");
} // end topo
```

How to find a vertex with no successors?

- Adjacency Matrix: A row with all zeros
- Adjacency List: array cell with an empty list (null entry)

How to delete a vertex?

Robert Lafore. 2002. Data Structures and Algorithms in Java (2 ed.). Sams, Indianapolis, IN, USA. Page 654

Topological Sorting using DFS

- We can use a similar idea as DFS to do topological sorting:
  - Start from any node, mark it as visited
  - Instead of printing the current node first(as in DFS), recursively call topological sorting on all its unvisited neighbors.
  - When the current node is fully explored (no more unvisited neighbors left), push it to a stack.
  - Continue until no more unvisited nodes left.

- At the end, stack contains the topological sort.

Shortest Paths Algorithm in a DAG

Pseudo-code for DAG shortest paths:

Initialize the tables/vectors d and parent

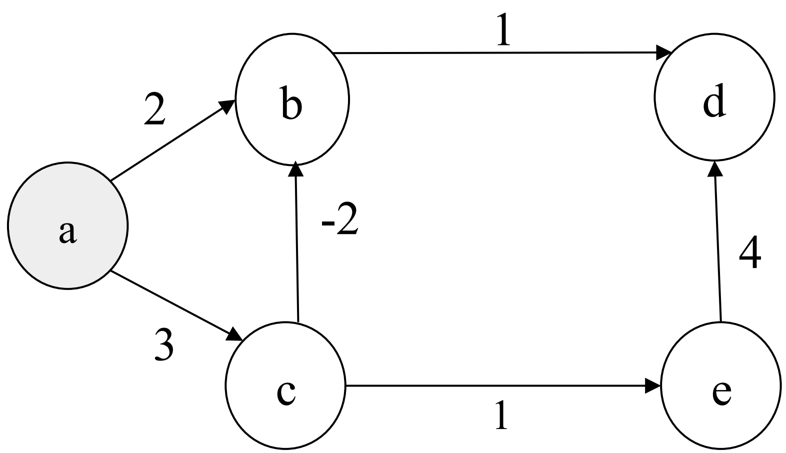
do topological sorting

for each node u taken from the topologically sorted order do

```
for each edge (u, v)  do
        if d[v] > d[u] + w(u, v)
                then  d[v] := d[u] + w(u, v)
                      parent[v] := u
```

Time complexity O(N+E)

Shortest Paths Algorithm in a DAG - Example



Topological sort: a, c, b, e, d

d[]

a	b	c	d	e

parent[]

a	b	c	d	e

All Shortest Path Algorithms

- Find all shortest paths from all nodes to all other nodes.
  - Floyd-Warshall Algorithm
    - It is based on *Dynamic Programming*

- Floyd-Warshall has O(N^3) time complexity, which is better than calling single source shortest path algorithm N times.

```
for (int k = 1; k =< N; k++)
   for (int i = 1; i =< N; i++)
      for (int j = 1; j =< N; j++)
         if ( ( d[i][k]+ d[k][j] ) < d[i][j] )
             d[i][j] = d[i][k]+ d[k][j];
```